Une image contenant texte, Police, Graphique, capture d’écran

Description générée automatiquement

Développement d’un chatbot pour la langue des signes

EDDY MBITCHA DAMEN et SEYFEDDINE MZALI

Table des matières

[INTRODUCTION 2](#_Toc172202748)

[I – Architecture 3](#_Toc172202749)

[1.1 – Application web 3](#_Toc172202750)

[a) Django 3](#_Toc172202751)

[b) Streamlit 4](#_Toc172202752)

[1.2 – Modèle et api 5](#_Toc172202753)

[a) API openai 5](#_Toc172202754)

[b) Modèle 6](#_Toc172202755)

[1.3 – Communication front et backend 11](#_Toc172202756)

[a) Envoie des données du front vers le backend 11](#_Toc172202757)

[b) Envoie des données du backend vers le front 12](#_Toc172202758)

[c) Schéma simplifié 13](#_Toc172202759)

[II – Difficultés et défis 13](#_Toc172202760)

[2.1 – Intégration du modèle et gestion de la caméra 13](#_Toc172202761)

[2.2 – Enregistrement du dataset 13](#_Toc172202762)

[2.3 – Organisation de l’équipe 13](#_Toc172202763)

[III – Axe d’amélioration 14](#_Toc172202764)

[3.1 – Affichage et design 14](#_Toc172202765)

[3.2 – Amélioration du modèle 14](#_Toc172202766)

[a) Augmentation des données 14](#_Toc172202767)

[b) Mauvaise qualité de la caméra 14](#_Toc172202768)

[c) Conception du modèle 14](#_Toc172202769)

[3.3 – Mise en ligne 15](#_Toc172202770)

[CONCLUSION 15](#_Toc172202771)

# INTRODUCTION

Dans le cadre de notre projet de fin d’année, nous avons décidé de développer une application web de type chatbot qui aurait une fonctionnalité de traduire la langue des signes.

La surdité affecte des millions de personnes dans le monde et elles utilisent souvent la langue des signes pour principal moyen de communication. Cependant, la méconnaissance de cette langue par la majorité de la population peut rendre difficile l’inclusion sociale et professionnelle de ces personnes.

L’objectif de notre projet vise à répondre à ce défi en développant un chatbot capable de traduire la langue des signes en texte écrit en temps réel dans le cadre d’une application chatbot.

Cet outils permettra de briser les barrières de communication en traduisant la langue des signes en texte.

Dans ce rapport nous verrons, l’architecture du projet, les difficultés et les défis rencontrés, les améliorations possibles et nous terminerons par une conclurons.

# I – Architecture

L’architecture de notre projet est assez complexe à comprendre. Elle est subdivisée en deux parties. Une partie application web qui elle-même se divise en 2 parties, le templates et le front qui interagissent entre elles constamment et une sous partie dédiée à l’entrainement du modèle.

## 1.1 – Application web

### a) Django

L'application web a été développée avec Django, un framework Python qui facilite la création sécurisée d'applications web. À la différence de la plupart des frameworks web, Django adopte le paradigme Modèle-Vue-Template (MVT) pour structurer les applications de manière claire et efficace.

Une image contenant capture d’écran, texte, diagramme

Description générée automatiquement

*Le modèle :* Dans cette architecture, le modèle est un fichier qui représente généralement la structure de la base de données de l'application. Cependant, dans notre cas spécifique, nous n'avons pas de base de données associée à l'application.

*La vue :* La vue est un fichier qui gère la logique de l'application en traitant les requêtes et les actions de l'utilisateur. Elle interagit également avec les templates pour présenter les informations à l'utilisateur et avec le modèle pour accéder aux données ou effectuer des opérations de traitement.

Dans notre cas, l’ensemble des fonctions qui gère l’interaction avec le modèle de deep learning sont dans le fichier ***view.py***

Les templates : constitue les fichiers HTML qui fournissent la mise en forme des pages en intégrant les données fournies par la vue. Dans notre cas ils contiennent des scripts Javascript qui vont intervenir dans la gestion de la logique d’affichage des données.

### b) Streamlit

Nous avons opté pour le Framework Streamlit afin de nous concentrer moins sur le développement frontend.

Streamlit est une solution open-source conçue pour simplifier la création d'applications web interactives et personnalisées, principalement dédiées à l'analyse de données et aux projets de machine Learning. Sa simplicité permet aux développeurs et aux data scientistes de convertir rapidement leurs scripts Python en applications web attrayantes et fonctionnelles, sans nécessiter de compétences avancées en développement web.

Cependant, nous avons rapidement abandonné l'utilisation de Streamlit car ce Framework ne permet pas une manipulation avancée de l'interface utilisateur. Nous l'avons donc utilisé principalement comme un outil pour tester l'API d'OpenAI.

## 1.2 – Modèle et api

### a) API openai

L'intégration de l'API OpenAI, en particulier ChatGPT, joue un rôle important dans notre projet. Notre modèle ne génère que des mots individuels les uns à la suite des autres, sans prendre en compte la structure grammaticale des phrases.

L'assemblage des mots générés peut mener à des phrases incohérentes ou mal structurées. ChatGPT reformule ces mots en phrases grammaticalement correctes puis les affiche en tant que suggestion en temps réel. Nous avons donc une **première phase** consistant à reformuler la phrase et ensuite une **deuxième phase** ou chatGPT permet de répondre à la phrase.

Phase 1 : restructuration du message

Une image contenant texte, capture d’écran, conception, illustration

Description générée automatiquement

*Etape 1 :* Le message provenant du modèle est récupéré au moment où l'utilisateur envoie son message.

*Etape 2 :* Le message passe dans une fonction ask\_openai qui effectue une requête vers l’api de chatGPT.

*Etape 3 :* En output le message structuré.

Une fois le message restructuré, il sera sous forme de suggestion de correction à l’utilisateur avant qu’il puisse envoyer son message.

Phase 2 : réponse du bot

Une image contenant capture d’écran, texte, conception, art

Description générée automatiquement

*Etape 1 :* On récupère le message sous forme structurée et on l’insère dans une fonction retournant la réponse du bot. La fonction va lire le message.

*Etape 2 :* Le bot reçoit le message dans la fonction et requête l’api openAI pour afficher la réponse du bot interlocuteur.

### b) Modèle

Le processus de création et d’entrainement du modèle s’est déroulé en plusieurs étapes.

#### b1) Collecte des images et création du dataset

La collecte des images a été une étape cruciale dans le développement de notre modèle car la performance dépendait de la qualité des données que nous avions utilisées.

Les datasets disponibles sur internet ne correspondaient pas exactement aux signes que nous recherchions. Ils étaient structurés de la manière suivante :

Une image contenant croquis, dessin, Dessin au trait, clipart

Description générée automatiquement

Il serait très complexe de former chaque mot en assemblant chaque lettre une par une. Face aux difficultés rencontrées pour trouver des datasets appropriés, nous avons pris la décision de créer notre propre dataset manuellement.

Ce dataset contient des mots entiers plutôt que des lettres individuelles. Chaque mot est associé à un signe, capturé sous différents angles pour éviter tout risque de surapprentissage.

Une image contenant dessin, texte, croquis, encre

Description générée automatiquement

La collecte des images a été effectuée à l'aide d'un script Python qui capture chaque mouvement de main et stocke les images dans des dossiers correspondants :

DATA\_DIR = './data'

if not os.path.exists(DATA\_DIR):

    os.makedirs(DATA\_DIR)

mots = ['IloveYou', "Hello", "No", "Yes"]

dataset\_size = 200

# Video capture a changer en fonction de la machine

cap = cv2.VideoCapture(1)

for i in mots:

    if not os.path.exists(os.path.join(DATA\_DIR, str(i))):

        os.makedirs(os.path.join(DATA\_DIR, str(i)))

    print('Collecte des données pour :', i)

    done = False

    while True:

        ret, frame = cap.read()

        cv2.putText(frame, 'Press "Z".', (100, 50),

                    cv2.FONT\_HERSHEY\_COMPLEX, 1.3, (0, 255, 0), 3, cv2.LINE\_AA)

        cv2.imshow('frame', frame)

        if cv2.waitKey(25) == ord('z'):

            break

    counter = 0

    while counter < dataset\_size:

        ret, frame = cap.read()

        cv2.imshow('frame', frame)

        cv2.waitKey(25)

        cv2.imwrite(os.path.join(DATA\_DIR, str(

            i), '{}.jpg'.format(counter)), frame)

        counter += 1

cap.release()

cv2.destroyAllWindows()

Dossier contenant la data :

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

L’étape suivante consiste à rendre utilisables ces données par un modèle de machine learning. Pour ce faire, nous avons conçu un second script qui utilise une librairie spécialisée dans l’estimation de la pose humaine à partie d’une vidéo ou d’une photo.

***Mediapipe*** permet d’extraire les points de repères des mains appelé **hand landmarks** à partir des images capturées avec **collect\_img.py**

Une image contenant plante, collage, personne

Description générée automatiquement

On compile ensuite ces informations et on les sauvegarde dans un fichier pickle qui servira à l’entrainement du modèle. Le fichier pickle contient des labels et des matrices de nombres correspondant à la position x et y de chaque landmarks.

Extrait de create\_dataset :

data = []

label = []

for i in os.listdir(DATA\_DIR):

    for j in os.listdir(os.path.join(DATA\_DIR, i)):

        # On va stocker toutes les informations de toutes les images

        data\_aux = []

        img = cv2.imread(os.path.join(DATA\_DIR, i, j))

        img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

        results = hands.process(img\_rgb)

        if results.multi\_hand\_landmarks:

            for hand\_landmarks in results.multi\_hand\_landmarks:

                for k in range(len(hand\_landmarks.landmark)):

                    # print(hand\_landmarks.landmark[i])

                    x = hand\_landmarks.landmark[k].x

                    y = hand\_landmarks.landmark[k].y

                    data\_aux.append(x)

                    data\_aux.append(y)

            data.append(data\_aux)

            label.append(i)  # contient les differents mots

#### b2) entrainement et test du modèle

La seconde phase du process de création consiste à entrainer le modèle. Pour cela nous avons créer un script qui utilise les données de points de repère des mains extraites par create\_dataset.py pour entrainer un modèle **RandomForest()** de classification.

Extrait de training.py :

data = np.asarray(informations['data'])

labels = np.asarray(informations['labels'])

x\_train, x\_test, y\_train, y\_test = train\_test\_split(

    data, labels, test\_size=0.2, shuffle=True, stratify=labels)

model = RandomForestClassifier()

model.fit(x\_train, y\_train)

y\_predict = model.predict(x\_test)

Avec le random Forest nous obtenons une accuracy de 99%.

Puis on sauvegarde le modèle :

f = open(f'sign\_detector\_model\_{current\_time}.p', 'wb')

pickle.dump({

    'model': model,

}, f)

f.close()

Maintenant vient la phase finale, le test du modèle.

Le test.py utilise le modèle de classification entrainé précédemment. On utilise openCV pour visualiser les résultats du modèle :

while True:

    data\_aux = []

    ret, frame = cap.read()

    if not ret:

        break

    frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

    results = hands.process(frame\_rgb)

    if results.multi\_hand\_landmarks:

        for hand\_landmarks in results.multi\_hand\_landmarks:

            mp\_drawing.draw\_landmarks(

                frame,

                hand\_landmarks,

                mp\_hands.HAND\_CONNECTIONS,

                mp\_drawing\_styles.get\_default\_hand\_landmarks\_style(),

                mp\_drawing\_styles.get\_default\_hand\_connections\_style()

            )

        for hand\_landmarks in results.multi\_hand\_landmarks:

            for k in range(len(hand\_landmarks.landmark)):

                x = hand\_landmarks.landmark[k].x

                y = hand\_landmarks.landmark[k].y

                data\_aux.append(x)

                data\_aux.append(y)

        prediction = model.predict([np.asarray(data\_aux)])

        predicted\_word = prediction[0]

        if isinstance(predicted\_word, str):

            current\_prediction = predicted\_word

        else:

            current\_prediction = label\_dict.get(int(predicted\_word), "Unknown")

        if current\_prediction != last\_prediction:

            print(f"Model prediction: {current\_prediction}")

            last\_prediction = current\_prediction

## 1.3 – Communication front et backend

L’application fonctionne grâce à une boucle d’interactions constantes entre le front et le backend. Pour détecter et retranscrire les signes directement dans la barre de message, le processus est divisé en deux parties : l’activation de la caméra par le front avec JavaScript, l’envoi des données vers le backend pour la prédiction des images en temps réel par le modèle, et enfin l’affichage de la prédiction dans le front.

### Envoie des données du front vers le backend

La première étape du processus consiste à activer la caméra puis d’envoyer les informations capturé par la caméra vers le backend pour que le modèle puisse prédire les images.

Cela est effectué par une fonction Javascript asynchrone ***getPrediction :***

async function getPrediction(imageData) {

        const response = await fetch('/predict/', {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json',

                'X-CSRFToken': document.querySelector('[name=csrfmiddlewaretoken]').value

            },

            body: JSON.stringify({ image: imageData })

        });

        const data = await response.json();

        console.log('Prediction:', data.prediction); // Log prediction to verify it's being received

        return data.prediction;

    }

La fonction envoie l’image capturée à l’endpoint Django ‘/predict’. Nous l’utilisons ensuite dans une autre fonction **captureAndPredict :**

async function captureAndPredict() {

        const imageData = captureImage();

        if (imageData) {

            const prediction = await getPrediction(imageData);

            if (prediction) {

                messageInput.value += ` ${prediction}`;

            }

        }

    }

Qui capture une image et obtient la prédiction puis l’ajoute dans le champ du message.

### Envoie des données du backend vers le front

Une fois les données récupérées du front, nous avons la classe PredictView qui va gérer les requêtes POST effectuées par les fonctions Javascript.

class PredictView(View):

    @method\_decorator(csrf\_exempt)

    def dispatch(self, request, \*args, \*\*kwargs):

        return super().dispatch(request, \*args, \*\*kwargs)

    def post(self, request, \*args, \*\*kwargs):

        try:

            data = json.loads(request.body)

#conversion de l’image au bon format pour quelle soit lu par le modèle

            image\_data = data['image']

            image\_data = base64.b64decode(image\_data.split(',')[1])

            np\_arr = np.frombuffer(image\_data, np.uint8)

            image = cv2.imdecode(np\_arr, cv2.IMREAD\_COLOR)

            prediction = get\_prediction\_from\_model(image)

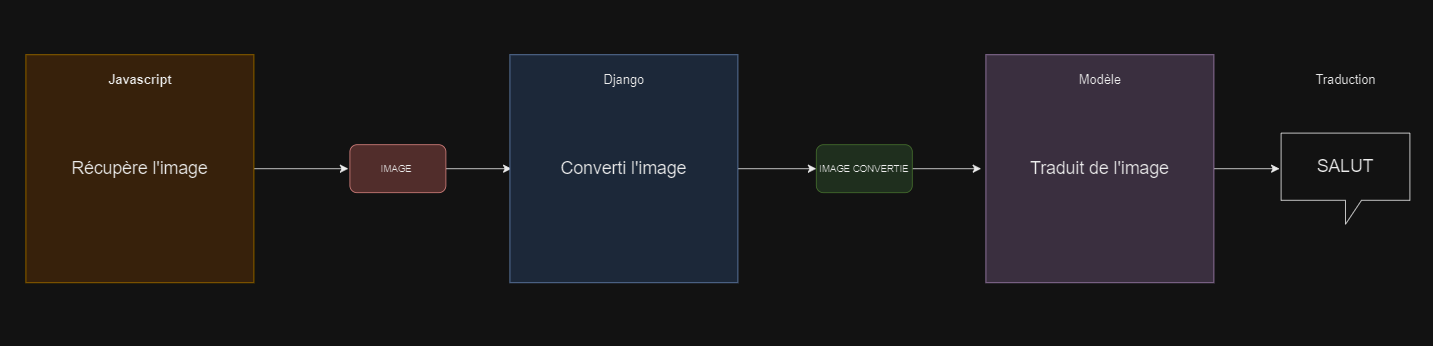
            return JsonResponse({'prediction': prediction}) # renvoie de la prediction dans l'async getPrediction

        except Exception as e:

            return JsonResponse({'error': str(e)}, status=500)

Les images reçues vont êtres décodées et la prédiction va etre effectuée en utilisant le modèle randomForest vu précédemment.

### Schéma simplifié



# II – Difficultés et défis

## 2.1 – Intégration du modèle et gestion de la caméra

La plus grande difficulté du projet était l'intégration du modèle à l'application Django. Cela a été une tâche extrêmement difficile en raison des défis techniques impliqués dans la conversion des réponses en JSON pour pouvoir interagir avec le frontend en JavaScript. Le processus était très complexe, nécessitait une bonne coordination entre plusieurs systèmes et surtout une bonne compréhension du langage Javascript et le fonctionnement des fonctions asynchrones.

Nous avons réussi à surmonter ce défis et atteindre notre objectif.

## 2.2 – Enregistrement du dataset

Le remplissage du dataset a été une étape assez laborieuse car il fallait enregistrer les données nous-mêmes. C'était une tâche qu'il fallait réaliser en une seule fois, car la structure de notre modèle ne permettait pas d'enregistrer un état du modèle pour compléter ensuite avec d'autres signes. Il fallait donc reprendre l'entraînement à zéro à chaque fois que nous oublions un mot ou faisions une erreur. Cependant, avec de la persévérance, de l'énergie et surtout beaucoup de concentration, nous avons pu enregistrer une cinquantaine de mots permettant de former des phrases simples.

## 2.3 – Organisation de l’équipe

Le rythme étant très intense au sein de cette formation, il a été très difficile de trouver un moment pour travailler sur le projet. Le développement du projet à fortement été retardé pendant le mois de juin à cause d’un arrêt maladie pour burnout.

Le projet était un projet assez robuste pour deux personnes. Il a fallu énormément travailler pour allier nos compétences afin de finir ce projet et le rendre fonctionnel.

# III – Axe d’amélioration

## 3.1 – Affichage et design

L’interface de l’application est assez rudimentaire. L’apparence d’une application web est quelque chose d’important et contribue à améliorer l’expérience utilisateur. On pourrait aussi améliorer la forme pour rendre l’interface encore plus responsive pour qu’elle puisse mieux s’adapter aux différents écran.

## 3.2 – Amélioration du modèle

### a) Augmentation des données

Le modèle ne comprend pour l’instant qu’une cinquantaine de mot et il arrive souvent qu’il fasse des erreurs et que l’utilisateur soit obligé de refaire les signes à chaque fois. Recueillir un plus grand nombre d’images avec divers gestes et angles pour agrandir la diversité de donner et permettre au modèle de mieux généraliser.

### b) Mauvaise qualité de la caméra

Les erreurs du modèle peuvent etre dû à la mauvaise caméra que nous avons utilisée pour récolter les images. Il faudrait effectuer des prétraitement des images avancés afin de réduire le bruit, normaliser etc…

### c) Conception du modèle

La source du problème peut aussi venir de la conception du modèle. Le modèle a été entrainé avec des procédés simples, nous n’avons pas utilisé de technique d’augmentation d’image comme la rotation, le changement de luminosité ou le flipping pour créer des variations. Il est fort probable que les erreurs du modèle viennent en parti du **surapprentissage**. Les techniques d’augmentation de donnée doivent etre utilisée prochainement si nous souhaitons avoir des résultats plus satisfaisant.

## 3.3 – Mise en ligne

La mise en ligne de l’application en tant que messagerie pourrait permettre de tester l’application en communiquant avec un interlocuteur humain et non un bot. Il faudrait donc mettre en place un système de connexion et de déconnexion des utilisateurs et créer une sécurité autour des mots de passe utilisateur.

# CONCLUSION

Pour conclure, ce projet a été une expérience enrichissante à bien des égards. De la conception initiale à l'implémentation finale, chaque étape nous a confrontés à des défis techniques et à prendre des décisions pour aboutir à un produit fonctionnel.

Nous avons débuté par la collecte de données, un processus laborieux mais indispensable pour garantir la qualité et la diversité des gestes pris en charge par notre modèle de reconnaissance. L'élaboration du modèle lui-même nous a amenés à explorer le domaine de la computer vision.

L'intégration de ce modèle dans une application web interactive a représenté un autre défi. L'architecture Django nous a permis de créer une interface utilisateur robuste et sécurisée, tandis que l'utilisation de Streamlit pour les premiers tests nous a permis d’apprendre à utiliser l’api de OpenAI.

Personnellement, ce projet m'a permis d’approfondir mes connaissances dans plusieurs domaines très différents en programmation pure, en traitement d'image, en développement web et j’ai pu apprendre la langue des signes. J’ai aussi perfectionné mes compétences en gestion de projet en sachant prendre les bonnes décisions et en résolution de problèmes techniques.

En résumé, cette expérience m'a permis de grandir en tant qu’informaticien et d’avoir confiance en mes capacités. Je suis assez fier du résultat obtenu meme si je pense que si nous avions eu un mois de plus, certains aspects auraient pu etre améliorés.